PREVENTION OF DEADLOCKS AND LIVELOCKS IN LOSSLESS, BACKPRESSURED PACKET NETWORKS

CROSS REFERENCE TO RELATED APPLICATION

This application claims priority of Provisional Application Serial No. 60/159147 which was filed on October 13, 1999.

FIELD OF THE INVENTION

The present invention relates generally to packet telecommunications networks, and in particular, to control of interconnected nodes in such packet networks in which deadlocks, livelocks and buffer overflow (packet loss) are avoided by providing a selective backpressure or feedback signal from a receiving node to a sending node having packets to send to the receiving node.

BACKGROUND OF THE INVENTION

Congestion occurs in packet networks when the demand exceeds the availability of network resources, leading to lower throughputs and higher delays. If congestion is not properly controlled, some packets will not get to their destinations, or will be unduly delayed. As a result, various applications requiring the information contained in the packets transported by the network may not meet their quality-of-service (QoS) requirements.

Proper congestion control is especially an important topic in emerging local area networks (LANs): large LANs with a heterogeneous mix of link speeds (ranging, for example, from 10 Mbps up to 1 Gbps) and the need to support the quality-of-service (QoS) requirements of multimedia applications from hundreds or even thousands of users.

When congestion builds up in a packet network, two general approaches are possible to cope with the shortage of buffer space. One approach is to drop incoming packets for which buffer is not available and to rely on the end-to-end protocols for the recovery of lost packets. Dropped packets are later retransmitted by end-to-end protocols. In many situations, dropping packets is wasteful, since the dropped packet has already traversed a portion of the network. Retransmissions of dropped packets also may lead to

unnecessarily large end-to-end delays.

An alternative approach is to insist that no packets should be dropped inside a packet network, even when congestion builds up. One way to accomplish this goal is to have the congested nodes send backpressure feedback to neighboring nodes, informing them of unavailability of buffering capacity and in effect stopping them from forwarding packets until enough buffer becomes available. While backpressure can be a useful approach, this method of dealing with congestion can potentially lead to deadlocks and livelocks in the network.

A deadlock is a condition under which the throughput of the network, or part of the network, goes to zero due to congestion (i.e., no packets are transmitted). This can be explained by reference to Fig. 1, which shows three interconnected nodes A, B and C. (Note that A, B and C can be any three adjacent nodes that form a "cycle" in a larger network of nodes.) Assume node A's buffer is full with packets destined for node B and beyond. Accordingly, node A sends a "stop" signal to its upstream nodes, in this example, node C. Likewise, if node B's buffer is full with packets destined for node C and beyond, and node C's buffer is full with packets destined for node A and beyond, both node B and C also send "stop" signals to their respective upstream nodes, in this example, node A and B. Under these circumstances, there is entire stoppage because each node in the loop has been directed to "stop" transmission. Stated another way, each network process, having resources required by the others, refuses or neglects to release them, causing the other processes to wait indefinitely.

In a livelock situation, the network is not stopped, but one or more individual packets are never transmitted. Fig. 2 shows a simple example of a livelock that can occur when a node 201 with separate high and low priority buffers 202 and 203, respectively, uses a strict-priority output link scheduling algorithm: low-priority packets in buffer 203 are forced to indefinitely wait in buffer 203 as new higher-priority packets continue to arrive at buffer 202 in node 201. These higher-priority packets get serviced, to the exclusion of packets in buffer 203. Even if all the link scheduling algorithms in a network are well-behaved (i.e., they don't indefinitely neglect the transmissions of any particular packets),

livelocks can still occur as a result of "network-level" effects. For instance, a hop-count-based flow control protocol might cause some node to (indefinitely) block its transmission of packets that have traveled fewer than a specified number of hops. From the foregoing, it is seen that livelocks are an extreme example of "unfairness," in which packets are "stuck" as they wait indefinitely in a queue while other packets (including new arrivals) are continually served.

Many different types of deadlocks have been identified and studied, along with methods of preventing them. (See, e.g., "System Deadlocks" by E. G. Coffman et al, *Computing Surveys*, Vol. 3, pp. 67-78, June 1971; "Some Deadlock Properties of Computer Systems" by R. C. Holt, *Computing Surveys*, Vol. 4, pp. 179-196, September 1972; "Deadlock Avoidance in Store-and-Forward Networks – I: Store-and-Forward Deadlock" by P. M. Merlin et al, *IEEE Trans. Commun.*, Vol. COM-28, pp. 345-354, March 1980; "Prevention of Deadlocks in Packet-Switched Data Transport Systems", by K. K. Gunther, *IEEE Trans. Commun.*, Vol. COM-29, pp. 512-524, April 1981; "Prevention of Store-and-Forward Deadlock in Computer Networks", by I. S. Gopal, *IEEE Trans. Commun.*, Vol. COM-33, pp. 1258-1264, December 1985; *Design and Validation of Computer Protocols* by G. J. Holzmann, Englewood Cliffs, NJ: Prentice Hall, 1991.)

In the LAN context, recent simulation results show that hop-by-hop backpressure can be better than TCP for dealing with short-lived congestion. (See "Selective Back-Pressure in Switched Ethernet LANs", by W. Noureddine et al., *IEEE GLOBECOM'99 Symposium on High Speed Networks*, Dec. 1999.) TCP, the dominant transport protocol in the Internet, uses packet drops as an indication of congestion and requires sufficient levels of loss in order to be an effective control. In a sense, TCP keeps increasing the load in order to increase the loss so that the necessary feedback signals are sent. Hop-by-hop backpressure helps reduce the number of packets dropped during periods of transient congestion, and avoids wasting the network resources so far already consumed up to this point. This is particularly important when we consider packets that might arrive at a LAN after traversing a wide area network (WAN). The penalty is that some links are "turned off" for short periods of time, perhaps negatively impacting other flows (i.e., those not

involved in the "congestion") as the backpressure propagates through the network. The advantage is that the distributed memory resources in a network can be used to buffer excess traffic generated by bursty sources. This helps bypass the costly TCP flow control mechanism, which may have a negative performance impact in the LAN context. (See "Flow Control in ATM networks: a survey" by S. Kamolphiwong et al., *Comp. Commun.*, Vol. 21, pp. 951-968, 1998.)

Although a backpressure congestion control mechanism, in which a node receiving a packet controls the stop-start behavior of a node intending to transmit a packet to the receiving node, can, in theory, eliminate packet loss in networks, this capability is obviously gone if packets need to be dropped to prevent deadlocks or to recover from a deadlock condition. Proper strategies for dealing with deadlocks increase in importance as data transmission rates increase to gigabit-per-second (and higher) rates. For a given network load level, the number of potential deadlocks per hour that have to be prevented, avoided, or recovered from, increases in proportion to the transmission rate. In addition, potential deadlocks will occur more frequently as the network loading increases. So, for instance, as gigabit Ethernet links are extended to cover greater distances in metropolitan-area networks (MANs) and WANs, heavier loading of long-distance links to make efficient use of the links will cause potential deadlocks to occur more frequently. Currently, proprietary hardware and high-quality fiber-optic lines can extend the distances between switches to greater than 70 km, permitting gigabit Ethernet implementations across MANs and even WANs. (See "Gigabit Ethernet ventures into the land beyond the Lan" by J. Caruso, *Network World*, p. 36, May 1999; and "Intelligent DWDM takes Gigabit Ethernet to the MAN" by N. Margalit, *Lightwave*, p. 101, June 1999.)

One simple way to deal with deadlocks is to just start dropping packets once a deadlock has occurred, or is "about to occur" (i.e., as congestion increases). For certain types of packets (for example, "real-time" traffic and traffic that can permit packet loss), this approach works fine. However, packets that must eventually reach their destinations will need to be retransmitted if they are dropped to avoid a deadlock or recover from a deadlock condition. The impact on total end-to-end delay, including interactions with

higher-layer protocols (such as TCP), needs to be considered. Also, retransmission of packets might even cause the "deadlock condition" to redevelop.

Another way to deal with deadlocks is to simply increase the size of available bandwidth and/or buffers. The obvious theory here is that if peak amounts of bandwidth and buffers are available and are dedicated for all network traffic flows, then buffers do not overflow and deadlocks are not created. This approach, however, is typically too wasteful of network resources and requires stringent admission control procedures.

Yet another way to deal with deadlocks is to use up/down routing on a spanning tree (see "Autonet: A High-Speed, Self Configuring Local Area Network Using Point-to-Point Links" by M. D. Schroeder et al., *IEEE Jour. Selected Areas Commun.*, Vol. 9, pp. 1318-1335, October 1991), thereby assigning directions to the links, and avoiding cycles. However, the selected paths (to create the spanning tree) are generally not the shortest and links near the root of the spanning tree become bottlenecks, limiting the network throughput.

Another way to avoid cycles in the network (which can lead to deadlocks) is to split each physical link into a number of virtual channels, each with its own queue and stop-start backpressure protocol. (See "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks" by W. J. Dally et al., *IEEE Trans. Comput.*, Vol. C-36, pp. 547-553, May 1987; and "Congestion Control in Asynchronous, High Speed Wormhole Routing Networks" by E. Leonardi et al., *IEEE Commun. Mag.*, pp. 58-69, November 1996.) Bandwidth is shared among the virtual channels. Layers of acyclic virtual networks and deadlock-free routes are created using the virtual channels. However, as the number of virtual channels increases, the scheduling becomes more complicated. Also, a method of associating individual packets with particular virtual channels is required.

Finally, more sophisticated buffer allocation strategies (structured buffer pools) can be used to prevent deadlocks. Extra buffers are allocated for packets that have higher "priority" because they have traveled greater distances (e.g., number of hops) in the network (or are closer to their destinations). In other words, using "distance" information in the packet headers, buffer space is reserved at each network node according to the

distance traveled through the network from a source (i.e., the number of hops). This technique is again complicated and expensive to implement.

There are several problems with these prior deadlock prevention techniques. First, and most important for the gigabit Ethernet scenario, they may not be compatible with the IEEE 802.3z standard. Most prior approaches require packet headers to include the "distance information" (such as, for example, the packet's hop count). There is no such provision for including distance information in the header of IEEE 802.3z packets. Alternatively, some other ("non-standard") way would need to be employed for transferring the "distance" information downstream to the next node.

A second problem with prior approaches is that although deadlocks are prevented, the end-to-end packet sequence may not be preserved. This may cause problems for some sessions that expect to receive packets in sequence, and perhaps can deal with "missing" (i.e., dropped) packets better than "out-of-sequence" packets. For instance, it is possible that a session's packets might be transmitted out-of-sequence to the next node because they were stored in different buffer classes (even though they all arrive with the same hop-count). Specific information would need to be kept regarding the order in which packets were stored in the various buffer classes.

A third problem with prior approaches is that although they resolve the deadlock problem, some do not eliminate the possibility of livelock. For example, if the scheduling algorithm and signaling protocol are not carefully designed, then in the continual presence of new arrivals with higher hop counts, packets in lower buffer classes might never have a chance to be transmitted.

A fourth problem with prior approaches is that network nodes need some type of "signaling message" to tell upstream neighbors which packets they can transmit (alternatively, they need a way to send negative acknowledgements when packets are dropped). Provision for such a signaling message does not exist in the IEEE 802.3z standard.

Finally, a fifth problem with some prior approaches is that a method of determining the current "distance to destination" information must be available in the network. This gets

particularly challenging when the possibility of network reconfigurations and routing table updates must be taken into account, which can change the source-to-destination distances.

SUMMARY OF THE INVENTION

The present invention is used in a packet communication network comprised of interconnected nodes arranged to transmit packets of variable length γ to adjacent nodes, where D is the maximum number of nodes that a packet must traverse through said network (i.e., the maximum number of hops) from an originating source to an ultimate destination. Each node in the network is, at the same time, both a transmitting node and a receiving node for its respective output and input links. For purposes of explanation of the present invention, the characteristics and arrangement of each node is best described by referring to an exemplary pair of nodes as a sending node X_ℓ connected to a receiving node R_ℓ via a link ℓ. Each node X_ℓ and R_ℓ includes a buffer for storing packets enroute from the originating source node to the ultimate destination node, and management hardware/software capability to (a) assign a local priority level λ_p (from amongst at least two possible priority levels) at the node to packets stored in the buffer, (b) formulate a feedback value f_ℓ sent from the receiving node R_ℓ to the sending node X_ℓ, that assures that there will be room in the buffer in the receiving node R_ℓ to store packets subsequently received from the upstream node X_ℓ, and (c) transmit from the sending node X_ℓ to the receiving node R_ℓ, only those packets in the buffer in the sending node X_ℓ that are eligible for transmission as a result of the fact that the packets have a priority level λ_p at X_ℓ that exceeds the feedback value f_ℓ received from the receiving node R_ℓ.

The priority level λ_p assigned to packets stored in the buffer at R_ℓ is based upon the ultimate destination to which the packets are to be transmitted, such that all packets intended for the same ultimate destination have the same priority level. Therefore, we represent the priority level associated with a particular destination d as the destination level λ^d. Initially, the priority level λ^d is set to 0 for all destinations. When a packet p with ultimate destination d arrives at R_ℓ from another network node (X_ℓ) over some link ℓ, the

priority level λ^d at R_ℓ associated with d is updated as the maximum of

(a) the prior value of λ^d at R_ℓ, or

(b) $1+f_\ell$

where f_ℓ is the value of the most recent transmit feedback sent over the reverse link ℓ' from node R_ℓ to node X_ℓ. Note that when a packet p with destination d enters the network at node n (over some network access link) the destination level λ^d at node n does not change. Note also that when the priority level λ^d at R_ℓ is increased to $1 + f_\ell$, the priority levels λ_p of all packets with ultimate destination d is automatically increased to $1 + f_\ell$, which is the new value of λ^d at R_ℓ.

In one embodiment of the present invention, the priority level assigning step is accomplished by assigning a priority level λ_p at R_ℓ that is less than or equal to D minus the number of hops remaining between the receiving node R_ℓ and the ultimate destination.

The feedback value f_ℓ sent from a receiving node R_ℓ to a sending node X_ℓ is determined by first setting in the buffer at the receiving node R_ℓ thresholds B_i that limit the maximum amount of space for packets with priority levels λ^d less than or equal to i. At all times, all B_i buffer threshold constraints must be satisfied. This division is not a physical partitioning of the buffer space, but is only an allocation of space. Allocation typically occurs when the system is initialized.

The receiving node R_ℓ thereafter monitors the priority levels λ^d of arriving and departing packets, and the increasing of priority levels λ_p of previously-stored packets, and thus keeps track of the total space in the buffer at R_ℓ occupied by packets of various priority levels λ^d. The feedback f_ℓ sent from the receiving node R_ℓ to the sending node X_ℓ represents the lowest priority level of packets that the receiving node R_ℓ could accept without violating any of the B_i buffer threshold constraints. In other words, the receiving node R_ℓ has room to accept packets of priority level $(1+f_\ell)$ or greater, without violating any of the buffer threshold constraints, but the receiving node R_ℓ cannot accept packets of

priority level f_ℓ or lower because it could possibly cause one or more of the buffer threshold constraints to be violated.

The present invention is a lossless method of preventing deadlocks and livelocks in backpressured packet networks. In contrast with prior approaches, the present invention does not introduce any packet losses, does not corrupt packet sequence, and does not require any changes to packet headers, such as attaching a hop counter, as required by the prior approaches. Because the proposed technique makes use of only the Destination Address in each packet header, and because the format of a typical gigabit-Ethernet packet, for example, contains the Destination Address but not a hop-counter field, the present invention can advantageously be used not only in a general packet network, but also in gigabit Ethernet (IEEE 802.3z) networks. In such networks, a PAUSE frame stops the flow of data frames for specified periods of time (indicated by a parameter in the PAUSE frame). A PAUSE frame that contains a parameter of zero time allows the flow of data frames to restart immediately.

BRIEF DESCRIPTION OF THE DRAWING

The present invention will be more fully appreciated from a consideration of the following Detailed Description, which should be read in light of the accompanying drawing in which:

Fig. 1 is a diagram illustrating three interconnected nodes that are involved in a deadlock condition;

Fig. 2 is a diagram illustrating a node having high and low priority buffers, in which a livelock condition may occur;

Fig. 3 is a diagram illustrating two interconnected nodes, and the forward data link and feedback control link connecting them;

Fig. 4 shows a network of three serially interconnected nodes, and, in accordance with the present invention, the "Level Table" associated with those nodes, showing the "level" assigned to packets buffered in those nodes based upon the destinations of the packets;

Figs. 5a and 5b show two instances of the same sample 10-node network, with

specific routing paths to reach destinations D_1 and D_2, respectively;

Fig. 6 is a diagram illustrating the crosspoint switch portion of a node arranged in accordance with the present invention, showing the various virtual queues within the switch;

Fig. 7 is a diagram illustrating the segmentation of the receive buffers within the switch of Fig. 6;

Fig. 8 is a diagram illustrating the buffer management parameters assigned, in accordance with the present invention, to the buffer of Fig. 7;

Fig. 9 is a flow chart illustrating the send functions performed at a sending node X_ℓ; and

Fig. 10 is a flow chart illustrating the receive functions performed at a receiving node R_ℓ.

DETAILED DESCRIPTION

Referring first to Fig. 3, consider a packet network that includes two nodes 301 and 302 connected by a one-way communication link ℓ. The sending node 301 is designated by X_ℓ, the receiving node 302 is designated by R_ℓ, and the communication link going in the reverse direction, from R_ℓ to X_ℓ, by ℓ'. Let S_ℓ denote the scheduling algorithm of link ℓ, i.e. the algorithm employed at node X_ℓ to select the next packet from those buffered at X_ℓ for transmission over ℓ. As in prior art arrangements, the scheduling algorithm S_ℓ could possibly base its selection on a number of factors, including packets' order of arrival, service priorities, service deadlines, and fairness considerations. The present invention enhances any prior art scheduling algorithm S_ℓ and avoids packet losses in the network by employing a selective backpressure congestion control mechanism for each link ℓ, to control the eligibility of packets to be transmitted over link ℓ. In this arrangement, before the buffer at the receiving node R_ℓ of a link ℓ overflows, a stop feedback is sent to the sending node X_ℓ, over the reverse link ℓ'. Unlike prior art backpressure mechanisms, the present invention is arranged to avoid the occurrence of deadlocks and livelocks in the network.

Under normal, uncongested conditions, all packets waiting at a sending node to be sent over link ℓ to R_ℓ, are eligible for transmission and may be selected by the link

scheduling algorithm S_ℓ. However, as the buffer at the receiving node R_ℓ gets congested, e.g., fills up close to its capacity, the set of packets that are eligible for transmission (i.e., those packets that may be selected by the scheduling algorithm S_ℓ to be sent over link ℓ) is gradually restricted. This is accomplished by sending a Transmit Feedback parameter f_ℓ, to be discussed shortly, over the communication link ℓ' in the reverse direction, from R_ℓ to X_ℓ. As the congestion at node R_ℓ subsides, the restriction placed on the transmission of packets over ℓ is gradually increased by designating more packets as eligible using a new value of Transmit Feedback f_ℓ.

Let D denote the maximum number of hops in any legitimate network route (or an upper bound on the number of hops if the maximum is, a priori, unknown). D depends on the network topology and routing protocol. For instance, if shortest path routing is used, then D is the diameter of the network. We associate with each packet p buffered at a node an integer value between 0 and D, inclusive; we call that assigned value the level of p and denote it as λ_p. When a packet is forwarded in the network from one node to another, no information about the level that was assigned to the packet in the first (sending) node is carried along with it. For instance, packet levels are not included in the packet headers (in contrast with, for example, prior deadlock-prevention schemes that carry hop counts in packet headers). To carry such information would require a change in existing Ethernet standards. Nonetheless, as will be discussed below, the present invention allows the receiving node to infer some partial information about the level that the packet held in the previous node. Typically, the level assigned to the packet in the new (receiving) node is different than its previous level.

Like packet levels, the Transmit Feedback f_ℓ also assumes an integer value between 0 and D, inclusive. The eligibility of packets for transmission over each link ℓ is determined by the value of the corresponding Transmit Feedback f_ℓ in accordance with the following rule:

Transmit Eligibility Rule:

A packet p waiting at node X_ℓ is eligible to be picked up by the scheduler of link ℓ for transmission over ℓ, if its current level λ_p satisfies

$$\lambda_p \geq f_\ell, \tag{1}$$

where f_ℓ is the most recent Transmit Feedback received by X_ℓ from the receiving node R_ℓ. It follows from this rule that when $f_\ell = 0$, all packets are eligible for transmission over ℓ. As f_ℓ increases, the protocol becomes gradually more restrictive and fewer and fewer packets are considered eligible for transmission over ℓ.

In a real network, it takes some time until a Transmit Feedback f_ℓ sent by R_ℓ reaches X_ℓ. Likewise, the effect of eligibility designations made at X_ℓ, as the result of the Transmit Feedback received from R_ℓ, does not reach R_ℓ until after a delay related to the propagation time of ℓ. In order to focus on the essence of the present invention, and not be sidetracked by secondary issues, we assume that the propagation delays of all network links are zero and that the Transmit Feedback generated at a receiving node R_ℓ is instantaneously sent to and detected by the sending node X_ℓ. Later, we will discuss necessary modifications in the protocol to accommodate real network scenarios where these conditions are not met.

Next, we describe how the level of a packet arriving at a node is determined. Let packet p arrive at node R_ℓ over link ℓ, and assume that f_ℓ is the most recent Transmit Feedback sent to X_ℓ. It follows that the level of p prior to transmission from the previous node (X_ℓ) must have been f_ℓ or larger. To guarantee freedom of deadlock in the network, it suffices to assign level $1 + f_\ell$ to packet p (as well as follow the buffer management and feedback rules described below). However, this simple approach can lead to the assignment of different levels, at a given node, to packets of the same session, which in turn can result in misordering of these packets when they are forwarded to the next downstream node.

To avoid misordering of packets belonging to the same session, it is advantageous

(but not essential) to adopt the following principle in the implementation of the present invention: at each node and at each point of time, all buffered packets that have a common destination should have the same level (so that all will be eligible/ineligible at the same times, and therefore selected for transmission in the correct order). This principle may be accomplished at each node by increasing the level of all packets with a common destination to the highest level among them (which also potentially increases their opportunities to be eligible for transmission). With this modification, it is more appropriate to view the level assignments at a node as being performed on a per-destination, rather than per-packet, basis. In accordance with this viewpoint, at a given node, let us denote the level associated with a destination *d*, by λ^d. The selective backpressure protocol of the present invention is based on the following rules for the assignment and updating of these destination levels.

Level Assignment Rules:

1. Every node *n* maintains a list of all destinations *d* that it encounters, along with the associated level λ^d. We refer to this list as the Level Table of node *n*. Initially, there are no entries in the Level Table.
2. By default, the level of any destination not included in the Level Table is zero. Accordingly, any destination which has a level equal to zero may be eliminated from the Level Table.
3. At any point of time, the level λ_p of each packet *p* which is buffered at node *n* is equal to the level λ^d associated with the corresponding destination *d*, at that time.
4. When a packet *p* with destination *d* arrives from another network node over some link ℓ, the level associated with *d* is updated as

$$\lambda^d \leftarrow \max\left(\lambda^d, [1+f_\ell]\right), \tag{2}$$

where f_ℓ is the value of the most recent Transmit Feedback sent over the reverse link ℓ'.

5. When a packet *p* with destination *d* enters the network at node *n* (over some network

access link) the level λ^d does not change. For instance, when a packet *p* arrives with destination *d* and that destination *d* is not included in the Level Table for node *n*, then the level λ^d remains set to zero.

6. When all buffered packets with destination *d* have left node *n*, it is permissible (but not necessary) to eliminate destination *d* from the Level Table at *n*. In other words, it is only necessary to keep values in the Level Table for destinations of currently-buffered packets. Note from rule 2 above that such elimination is equivalent to resetting λ^d to zero. It also serves to automatically "refresh" entries in the Level Table, which is needed since the topology or routing paths may change over time.

Before proceeding, we present a few important observations regarding the above rules:

- If all packets encountered by node *n* and destined for destination *d* enter the network at *n*, then λ^d is always equal to zero since it is never subjected to the update in rule 4 above. This means that packets arriving into the network at node *n* and destined for *d* will assume level zero at *n*, provided that *n* does not encounter any traffic destined for *d* that comes from another network node.
- When a packet arrives at node *n* from another network node, it will be assigned with a level of at least 1, since the level associated with its destination will undergo the update in rule 4 above.
- Updating according to the above rules will never result in a level larger than D. By the time the level associated with a packet reaches D, the packet must have reached its destination. Typically, packets' levels are less than D when they reach their destinations.
- Nodes do not need to keep track of the levels associated with each and every individual buffered packet. Each node only needs maintain a short Level Table listing the destinations for which packets are buffered at the node.

To illustrate how levels are assigned, Fig. 4 shows a network example of various levels and Transmit Feedback values. Three serially interconnected nodes A, B and C are

shown, with the heavy line from left to right indicating the packet data path, and the lighter line from right to left indicating the feedback path. Nodes X, Y and Z are assumed to be destination nodes that are interconnected with the nodes A, B and C by yet other parts of the network that are not shown. In other words, we are assuming for illustration purposes that a routing process will have certain packets destined for nodes X, Y and Z pass through nodes A, B and C in order to get to their final destinations.

As shown in Fig. 4, node A has permission to transmit a packet destined to node X to node B, because the level associated with X in node A's level table (i.e., "1") is not less than the Transmit Feedback parameter (i.e., "0") returned to node A via the reverse link to node B. When the same packet reaches and is buffered at node B, the level of the packet is increased to "4", which is the value associated with destination X in node B's Level Table. This is because of Rule 4 above. Similarly, node B has permission to transmit the packet destined to node X to node C, because the level associated with X (i.e., "4") is not less than the Transmit Feedback parameter (i.e., "1") received in node B via the reverse link from node C. When the packet reaches and is buffered at node C, the entry for destination X in node C's level Table was "1". This value, however, is increased to "2" (which automatically increases the level to "2" of any other packets in the node C buffer destined for node X), again as a result of the application of Rule 4 above. Notice in this simple example that the level of the packet destined for node X went from "1" to "4" to "2" as it passed from nodes A to B to C.

The example of Fig. 4 shows that a packet's level is quite "volatile" as it travels through a network; the level can even increase while it is buffered at a node (as at node C in Fig. 4). One of the few things that can be said about a packet's level is that it is less than or equal to D minus the number of hops remaining to the packet's destination. As an illustration, Figs. 5a and 5b show two instances of the same sample 10-node network, with specific (exemplary) routing paths to reach destinations D_1 and D_2, respectively. Inside each node is an integer that indicates the maximum possible level of packets destined to D_1 or D_2. Consider Fig. 5a first. Note that every node 501 –510 has some path to reach destination D_1, which is node 510. At a leaf of the routing tree (e.g., nodes 501 and 504),

the level is always zero. A packet entering the network at node 507, however, might be assigned a level as large as "3" because of the impact on node 507's Level Table from packets that entered the network at a previous node, such as node 501 and that may have attained a level equal to 3 at node 507. Likewise, the packets that enter at node 504 (with their levels initially set to "0") might have their levels increased to "3" when they reach node 507. Similar observations can be made with the set of paths in Fig. 5b that route packets to node 501 (which is labeled as D_2 in Fig. 5b). In this example, nodes 506, 509 and 510 are leaf nodes, and a packet going from node 506 to node 502 would have its level increased from "0" to "4" as it travels toward node 501.

In summary, a packet's level increases and decreases as a function of the topology, the distance the packet travels, the "recent history" of arrivals (throughout the network), and the state of network congestion.

Now that we have described the concepts of packet (or destination) levels, the Transmit Feedback, and the Transmit Eligibility Rule, we explain in the next few paragraphs how the values for the Transmit Feedback are selected. First, though, we need to briefly describe the general switch architecture under consideration.

The selective backpressure technique of the present invention can be used in networks with arbitrary switch configurations. So, we consider a general switch model for each node of the network, as illustrated in Fig. 6. The switch includes a cross point matrix selectively interconnecting N input links 601-1 through 601-N with M output links 610-1 through 610-M. A virtual input-output queue 620-1,1 through 620-N,M is associated with each input-output pair (i,j), a virtual Receiving Queue 630-1 through 630-N is associated with each incoming network link, and a virtual Sending Queue 640-1 through 640-M is associated with each outgoing network link. The Receiving Queue 630 associated with an incoming link ℓ is used for determining the Transmit Feedback f_ℓ , and the Sending Queue 640 is used with its associated scheduling algorithm in determining the next packet to be transmitted - selecting from those packets that are eligible, including those generated at the node and stored in the buffers assigned for traffic entering the network at the node. Note here that Fig. 6 does not explicitly show the buffers assigned for the traffic entering (and

also leaving) the network at the node (i.e., over some network access link). Note also that in the switch model illustrated in Fig. 6, each packet is at the same time treated as belonging to both a Receiving and a Sending Queue. We assume that the scheduling algorithm S_ℓ is well-behaved, meaning that eligible packets in each Receiving Queue (i.e., arriving from each of the input links) will eventually be selected for transmission over an output link. Otherwise a livelock would result if the link scheduling algorithm S_ℓ continually refused to select a particular eligible packet for transmission over an output link.

This general switch model of Fig. 6 can be physically implemented in many ways, which is why it is important to state that the defined input-output queues 620, the Receiving Queues 630, and Sending Queues 640 are virtual. For instance, in a completely-shared-memory switch, all Input-Qutput, Receiving, and Sending Queues are maintained in lists as packets arrive on various incoming links and depart on various outgoing links. In an input-buffered (output-buffered) switch, however, the Receiving (Sending) Queue could be a physical buffer and the other Queues would still be virtual entities that are maintained by, for example, keeping lists of packets destined for (coming from) various outgoing (incoming) links.

Now consider again an arbitrary network link ℓ and the receiving node R_ℓ. We refer to the input buffer at R_ℓ that is associated with the incoming link ℓ as the Receiving Queue of ℓ. Let the size of this buffer be denoted by b^ℓ, and let γ_{max} denote the maximum size of a packet in the network. Our core idea for deadlock prevention is to manage the Receiving Queue of each link ℓ and to set the value of the Transmit Feedback f_ℓ as described in the following paragraphs.

As illustrated in Fig. 7, we divide a buffer 701 having a total buffer size b^ℓ into D parts b_i, i=1,2,...,D satisfying

$$b_i \geq \gamma_{max}, \tag{3}$$

$$b^\ell = \sum_{j=1}^{D} b_j \geq D \cdot \gamma_{max}. \tag{4}$$

This division is not a physical partitioning of the buffer space; it is only an *allocation* of

space. Note here that in reality, the size b' of buffer 701 is only weakly dependent on the maximum route length D. Most of the buffer space is in b_1 (i.e., typically $b_j < b_1$ for $j > 1$) and the partitioning is "virtual."

We refer to b_i as the buffer budget of level i and require that a packet of level i be accepted into buffer 701 only if there is enough budget available for it at levels i or below. Let n_i, $i = 1, 2, \ldots, D$ denote the combined size of packets of level i that are stored in the Receiving Queue of link ℓ. The above requirement may be stated as

$$n_1 \le b_1, \tag{5}$$

$$n_1 + n_2 \le b_1 + b_2, \tag{6}$$

or, more generally

$$\sum_{j=1}^{j=i} n_j \le \sum_{j=1}^{j=i} b_j, \quad i = 1, 2, \cdots D. \tag{7}$$

Equivalently, we may define buffer threshold constraints $B_i = \sum_{j=1}^{j=i} b_j$ that limit the maximum buffer capacity that can be occupied by packets of level less than or equal to i. At all times, all B_i buffer threshold constraints must be satisfied (as in Equation (7)). Note that $B_D = b'$, the size of the buffer.

In order to observe the above requirements (to prevent deadlocks and livelocks), it is not necessary to physically partition the Receiving Queue into different segments. Instead, as illustrated in Fig. 8, which is another view of buffer 701 of Fig. 7, the above requirement may be implemented using a set of buffer management parameters m_i defined as

$$m_i \triangleq \sum_{j=1}^{j=i} (b_j - n_j), \qquad i = 1, 2, \cdots, D. \tag{8}$$

Equivalently, $m_i = B_i - \sum_{j=1}^{j=i} n_j$. m_i refers to that part of the combined buffer budget of levels $j \le i$, which is not allocated to packets of levels $j \le i$. This means that out of the combined buffer budget of levels $j \le i$, a budget m_i is either allocated to packets of levels $j > i$ or not

allocated to any packets at all. With this notation, m_D equals the total size of the vacant space in the buffer. Notice that since packets of level *j* can use the buffer budget of any level $k \leq j$, the term $b_j - n_j$ in (8) can be negative, for some *j*. However, m_i cannot be negative for any *i* since packets of levels $j \leq i$ cannot use the buffer budget of a level higher than *i*.

It follows from (8) that when a new packet of length γ and level *j* is stored in the buffer (leaves the buffer), m_i must be decreased (increased) by γ for all $i \geq j$. Similarly, when the packet's level is increased from *j* to *k*, then according to (8), m_i must be increased by γ for all levels *i*, $k > i \geq j$.

The above results also provide the guideline for choosing the Transmit Feedback f_ℓ to be sent over the reverse link ℓ' to the upstream node. Since the parameters m_i should always be nonnegative, the buffer can store a new packet of level *j* and arbitrary length, provided that currently $m_i \geq \gamma_{max}$ for all $i \geq j$. On the other hand, when a packet arrives following the sending of the Transmit Feedback f_ℓ, the level assigned to it could be as low as $1 + f_\ell$. Since we would like to set the value of the Transmit Feedback f_ℓ as low as possible, we conclude that f_ℓ should be set to a level *j* such that $m_i \geq \gamma_{max}$ for all $i \geq j+1$, and $m_j < \gamma_{max}$. It follows that f_ℓ should be set to the highest level *j* for which $m_j < \gamma_{max}$. Accordingly, if $m_i \geq \gamma_{max}$ for all $i = 1, 2, \ldots D$, then we set $f_\ell = 0$, allowing the transmission of packets of any level, over link ℓ. Conversely, if $m_D < \gamma_{max}$, then it follows that $f_\ell = D$. As we said before, if there is any packet of level D in the upstream node, it must be destined for that node itself. Therefore, all packets waiting to be sent over link ℓ must have a level less than D. It follows that, with the Transmit Feedback set to D, no packet is eligible to be sent over ℓ.

We now summarize the rules to be followed in accordance with the present invention for feedback setting of link ℓ and management of its Receiving Queue.

Buffer Management Rules:

- When the Receiving Queue of link ℓ is empty, initiate

$$m_i = \sum_{j=1}^{j=i} b_j, \qquad i = 1, 2, \cdots, D. \tag{9}$$

- If a packet of length γ arrives and is buffered at level j, decrease m_i by γ, for $i \geq j$.
- If a packet of length γ and level j leaves the buffer, increase m_i by γ, for $i \geq j$.
- If a packet of length γ is increased from level j to level $k > j$, increase m_i by γ, for all i, such that $k > i \geq j$.

Transmit Feedback Rule:

Set link ℓ 's Transmit Feedback $f_\ell = j$, where j is the largest level for which $m_j < \gamma_{max}$. If no such level exists, set $f_\ell = 0$.

Finally, it is to be noted that, as the result of applying the selective backpressure technique in accordance with the present invention, by selectively designating packets as eligible for transmission, the present invention improves the normal operation of scheduler S_ℓ of each link ℓ. This improvement is obtained because, through the use of our eligibility control mechanism, the order in which packets waiting for transmission are in fact transmitted, is changed from the order that would otherwise occur through the operation of scheduler S_ℓ. This is in contrast to the performance of a plain backpressure mechanism, which does not change the order of packet transmissions over a link, since either all packets waiting at a link are eligible for transmission or no packet may be sent at all.

Fig. 9 is a flow chart illustrating the send functions performed at a sending node X_ℓ. In step 901, a determination is made as to which packets are eligible for transmission over link ℓ, by determining if the priority level for a packet is greater or equal to the feedback level, or $\lambda_p \geq f_\ell$, as in Equation 1. Any arbitrary scheduling algorithm S_ℓ is then used to select the next packet for transmission from among those that are eligible, in step 903, whereupon the process of Fig. 9 returns and repeats step 901.

Fig. 10 is a flow chart illustrating the receive functions performed at a receiving node R_ℓ. In step, 1001, the buffer threshold constraints B_i are initialized. Then, in step 1003, the priority level λ^d of packets destined for destination d is updated. Next, in step 1005, the counters n_i that track the total buffer space occupied by packets with the priority

levels i, are updated. Note here that at most two counters n_i can possibly change: if the incoming packet results in the priority level λ^d being raised to $1+f_\ell$, then the counters n_i associated with both the previous (prior to arrival of the incoming packet) and new priority levels are changed; alternatively, if $1+f_\ell$ is less than or equal to the previous priority level λ^d, then only the counter n_i associated with that priority level is changed.

In step 1007, a new feedback signal f_ℓ is sent over reverse link ℓ', if necessary, i.e., if the status of any buffers have changed such that the feedback signal has changed in accordance with the requirement that the feedback f_ℓ sent from the receiving node R_ℓ to the sending node X_ℓ represents the lowest priority level of packets that the receiving node R_ℓ could accept without violating any of the B_i buffer threshold constraints. In other words, the feedback value must be set such that the receiving node R_ℓ will have room to accept packets of priority level $(1+f_\ell)$ or greater, without violating any of the buffer threshold constraints. The process then returns to step 1003, where the process is repeated when subsequent packets are received.

As indicated previously, one major advantage of the present invention is that the sending of the Transmit Feedback signals to the upstream neighbor is easily implemented in currently available IEEE 802.3 gigabit Ethernet, using the standard PAUSE frames. The invention can also be used in other networks that use different methods to signal congestion status to upstream nodes. Rather than coding the PAUSE frame's parameter to represent the period of time that the upstream neighbor should not send data frames, the parameter is coded to represent the various Transmit Feedback values.

Various alternatives and extensions of the present invention are possible. For example, up to this point, we have ignored the effects of propagation delays. The basic modification needed to incorporate propagation delays into the present invention is to make sure that nodes transmit their feedback signals with enough "lead time" so that the controls "take effect" at the appropriate moments (assuming worst-case conditions). For example, if a node determines that for a given incoming link a particular Transmit Feedback signal needs to take effect at time t_0, then it will need to transmit the signal at time $t_0 - T$, where T is the round-trip propagation time of that link. As the buffer occupancy continues to

change, note that a node might need to transmit an updated Transmit Feedback between the time t_0 - T and the time t_0.

Specifically, very little needs to change to incorporate the effects of a round-trip delay T in the present invention. The Transmit Eligibility Rule and the Buffer Management Rules are unchanged. Only the Transmit Feedback Rule and the Level Assignment Rules need to be slightly modified. First, in determining what Transmit Feedback to send at time t_0, the receiving node R_ℓ needs to consider the "worst-case" (maximum) occupancy of its Receiving Queues at time t_0 + T in the future. If the link rate from X_ℓ to R_ℓ is r, then the occupancy of a Receiving Queue may increase by at most rT. Consequently, the modified rule is:

Transmit Feedback Rule:

Set link ℓ 's Transmit Feedback $f_\ell = j$, where j is the largest level for which m_j - rT $< \gamma_{max}$. If no such level exists, set $f_\ell = 0$.

If desired, more complex "book-keeping" algorithms can also be incorporated to make more efficient use of memory and bandwidth resources. Using techniques similar to those proposed in "A New Feedback Congestion Control Policy for Long Propagation Delays", by I. Iliadis, *IEEE J. Select. Areas Commun.*, Vol. 13, pp. 1284-1295, September 1995, and information about the recent history (from time t_0 - T to time t_0) of feedback signals that were transmitted on a link to an upstream neighbor can be used to compute an upper bound (perhaps, less than rT) on the amount of information that will reach the downstream node between time t_0 and time t_0 + T.

Second, the (modified) Level Assignment Rule is based on the current buffer occupancy (just as in the T = 0 special case considered previously):

Level Assignment Rule:

When a packet p with destination d arrives from another network node over some link ℓ, the level associated with d is updated as

$$\lambda^d \leftarrow \max(\lambda^d, 1+j), \tag{10}$$

where *j* is largest level for which currently $m_j < \gamma_{max}$ (if no such level exists, set j =0). Note that because we make worst-case assumptions in the (modified) Transmit Feedback Rule, we can assign levels to packets at time t_0 without needing to remember what Transmit Feedback value was sent at time t_0 - T. Note that the Transmit Feedback value j* at time t_0 is greater than or equal to the *j* value in the Level Assignment Rule at time t_0 + T. This greatly simplifies the implementation.

What happens if the receiving node R_ℓ has an incorrect estimate of the round-trip propagation delay T? If the actual delay is greater than the estimate, then some packet loss may occur and deadlocks are possible. On the other hand, if the actual propagation delay is less than the estimated value used by node R_ℓ, then the only penalty is a (slight) loss in efficiency and perhaps some wasted buffer. Consequently, in local and metropolitan area network applications where link propagation delays are small, it might be preferable to select some upper bound on link propagation delays of the network as the parameter T used on all links - just as there are maximum distances for links due to physical constraints. If the protocol is used in wide area networks, however, it is probably better to individually set the T parameter for each link. Finally to guarantee no packet loss in the presence of nonzero propagation delays, the buffer size must be increased such that $b_1 \geq \gamma_{max} + rT$. Equation (3), constraining b_i, still holds for $i = 2, 3, \ldots, D$. Another option in dealing with long-propagation environments is to reinterpret some of the Transmit Feedback values (most likely the higher values) as permission to transmit up to a specified maximum number of packets of specified levels, rather than an unlimited number of packets. Additional Transmit Feedback messages would need to be sent to grant permission to transmit additional packets.

Before briefly presenting a few other possible extensions of the present invention, it is important at this point to mention the relationship of the invention with end-to-end congestion control techniques. While the present invention addresses the problems of short-term congestion overflows and deadlocks, it does not address the issue of fairness in providing service to different users. One way of resolving this shortcoming is to couple this

technique with end-to-end congestion control schemes that handle congestion problems on a quasi-static basis while providing the desired fairness and/or priorities in the amount of services given to different users in the long run. This can be accomplished in a number of ways that will be apparent to persons skilled in the art. (See, for example, "A Class of End-to-End Congestion Control Algorithms for the Internet" by S. J. Golestani et al., *Proc. 6th International Conf. On Network Protocols*, October 1998.)

Finally, we mention a few more possible extensions of the present invention. First, it is sometimes advantageous to incorporate some packet dropping to address other network issues, such as aging of packets due to errors, and blocking caused when certain network links are "overloaded". (See "A Simple Technique that Prevents Packet Loss and Deadlocks in Gigabit Ethernet", by M. Karol et al, *Proc. 1999 International Symposium on Communications (ISCOM'99)*, pp. 26-30, November 1999.) Second, if it is desired to internetwork the proposed lossless network with a network (e.g., TCP) that depends on losses to rate control its sources, then it is simple to design a "gateway" between the two networks. For instance, an edge device near a TCP source, or near the WAN, could be used to convert the lossless LAN's end-to-end technique to the TCP "loss technique" (i.e., by dropping packets) to implicitly rate control the TCP sources. Third, the present invention can be modified such that some Classes of Service (CoS) will be allowed to "ignore" the Transmit Feedback congestion control signals that gradually restrict the set of packets eligible for transmission. This is possible if, perhaps, there are dedicated buffers set aside for them. Alternatively, all nodes might be programmed to always treat particular CoS packets to be of no less than a certain minimum level.